日　本　国　特　許　庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出願年月日
Date of Application: ２００２年　７月２２日

出願番号
Application Number: 特願２００２－２１３００５

[ST.10/C]: [ＪＰ２００２－２１３００５]

出願人
Applicant(s): オークマ株式会社

２００３年　４月２５日

特許庁長官
Commissioner,
Japan Patent Office 太田信一郎

出証番号　出証特２００３－３０３０１００

【書類名】　特許願

【整理番号】　3M1-0537

【提出日】　平成14年 7月22日

【あて先】　特許庁長官殿

【国際特許分類】　G06F 11/00

【発明者】

【住所又は居所】　愛知県丹羽郡大口町下小口５丁目２５番地の１　オークマ株式会社大口工場内

【氏名】　大山　博司

【発明者】

【住所又は居所】　愛知県豊橋市北山町東浦２－１　高師住宅４－４０３

【氏名】　高田　広章

【発明者】

【住所又は居所】　愛知県丹羽郡大口町下小口５丁目２５番地の１　オークマ株式会社大口工場内

【氏名】　福島　盛雄

【特許出願人】

【識別番号】　000149066

【氏名又は名称】　オークマ株式会社

【代理人】

【識別番号】　100075258

【弁理士】

【氏名又は名称】　吉田　研二

【電話番号】　0422-21-2340

【選任した代理人】

【識別番号】　100096976

【弁理士】

【氏名又は名称】　石田　純

【電話番号】　0422-21-2340

【手数料の表示】

【予納台帳番号】　001753

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　論理回路のインターフェース方法およびインターフェースを備えた装置

【特許請求の範囲】

【請求項１】　論理演算素子により構成される論理回路において、インターフェース定義言語を用いてインターフェースを定義し、前記インターフェース定義言語はソフトウェアオブジェクトに対するインターフェース定義言語と共通部分を有し、前記インターフェース定義言語は機能ごとに機能名および該機能に対する引数および戻り値を宣言する手段を具備し、前記インターフェースを実現するサーバーインターフェース回路には前記インターフェース定義言語に記述された前記機能名を識別する入力および引数の入出力を行う手段および戻り値の出力を行う手段のうち少なくとも前記機能名を識別する入力を行う手段を具備させることを特徴とする論理回路のインターフェース方法。

【請求項２】　論理演算素子により構成される論理回路において、インターフェース定義言語を用いてインターフェースを定義し、前記インターフェース定義言語はソフトウェアオブジェクトに対するインターフェース定義言語と共通部分を有し、前記インターフェース定義言語は機能ごとに機能名および該機能に対する引数および戻り値を宣言する手段を具備し、前記インターフェースを実現するクライアントインターフェース回路には前記インターフェース定義言語に記述された前記機能名を識別する出力および引数の出入力を行う手段および戻り値の入力を行う手段のうち少なくとも前記機能名を識別する出力を行う手段を具備させることを特徴とする論理回路のインターフェース方法。

【請求項３】　論理演算素子により構成される論理回路において、インターフェース定義言語を用いてインターフェースを定義し、前記インターフェース定義言語はソフトウェアオブジェクトに対するインターフェース定義言語と共通部分を有し、前記インターフェース定義言語は機能ごとに機能名および該機能に対する引数および戻り値を宣言する手段を具備し、

前記インターフェースを実現するサーバーインターフェース回路として前記インターフェース定義言語に記述された前記機能名を識別する入力および引数の入

出力を行う手段および戻り値の出力を行う手段のうち少なくとも前記機能名を識別する入力を行う手段を具備したサーバー側論理回路と、前記インターフェースを実現するクライアントインターフェース回路として前記機能名を識別する出力および引数の出入力を行う手段および戻り値の入力を行う手段のうち少なくとも前記機能名を識別する出力を行う手段を具備するクライアント側論理回路とを備え、

前記サーバー側論理回路と前記クライアント側論理回路とに関し、前記機能名を識別する出力を行う手段から前記機能名を識別する入力を行う手段へのデータ転送を可能とし、且つ、前記引数を入出力する手段と前記引数を出入力する手段とが互いに備わっていれば前記引数を入出力する手段と前記引数を出入力する手段とのデータ転送を可能とし、且つ、前記戻り値を出力する手段と前記戻り値を入力する手段とが互いに備わっていれば前記戻り値を出力する手段から前記戻り値を入力する手段へのデータ転送を可能とする論理回路を合成することを可能とすることを特徴とする論理回路のインターフェース方法。

【請求項４】　論理演算素子により構成される論理回路において、インターフェース定義言語を用いてインターフェースを定義し、前記インターフェース定義言語はソフトウェアオブジェクトに対するインターフェース定義言語と共通部分を有し、前記インターフェース定義言語は機能ごとに機能名および該機能に対する引数および戻り値を宣言する手段を具備し、前記インターフェースを実現するサーバーインターフェース回路には前記インターフェース定義言語に記述された前記機能名を識別する入力および引数の入出力を行う手段および戻り値の出力を行う手段のうち、少なくとも前記機能名を識別する入力を行う手段を具備することを特徴とするインターフェースを備えた装置。

【請求項５】　論理回路において、インターフェース定義言語を用いてインターフェースを定義し、前記インターフェース定義言語はソフトウェアオブジェクトに対するインターフェース定義言語と共通部分を有し、前記インターフェース定義言語は機能ごとに機能名および該機能に対する引数および戻り値を宣言する手段を具備し、前記インターフェースを実現するクライアントインターフェース回路には前記インターフェース定義言語に記述された前記機能名を識別する出

力および引数の入出力を行う手段および戻り値の入力を行う手段のうち、少なくとも前記機能名を識別する出力を行う手段を具備することを特徴とするインターフェースを備えた装置。

【請求項６】　請求項５に記載のインターフェースを備えた装置において、前記クライアントインターフェース回路の接続端子は、前記サーバーインターフェース回路の接続端子とシステムバスとのいずれとも接続可能とし、前記サーバーインターフェース回路の接続端子へ接続した場合には前記サーバーインターフェース回路を介して装置を駆動可能とし、システムバスに接続した場合には中央処理装置は前記クライアントインターフェース内部のレジスタの値を読み書き可能とし、前期中央処理装置は前記サーバーインターフェース回路を介する装置の機能を代替することを可能とするインターフェースを具備することを特徴とするインターフェースを備えた装置。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、ＡＮＤ／ＯＲ／ＮＯＴ等の各種ゲートやフリップフロップ等の論理演算素子により構成される論理回路のインターフェース方法およびインターフェースを備えた装置に関し、特に、ソフトウェアオブジェクトとハードウェアオブジェクト（装置）の扱いを統一可能とするものである。

【０００２】

【従来の技術】

従来、マイクロプロセッサと周辺ＬＳＩとのインターフェースにおいて、状態レジスタやコマンドレジスタなどと呼ばれるレジスタを介して、状態値を読み出したり、所定の処理の実行を制御したりするインターフェース方法が用いられている。

【０００３】

このため、周辺ＬＳＩを制御するプログラムを設計するプログラマーは、周辺ＬＳＩのインターフェースを理解するために、レジスタの意味や複数のレジスタ間の関連性を理解し、レジスタの設定手順に留意してプログラムを作成する必要

がある。

【０００４】

【発明が解決しようとする課題】

ところで、近年の周辺ＬＳＩは、その機能が複雑化しており、多数の状態レジスタやコマンドレジスタを備えている。

【０００５】

従って、レジスタの意味や関連性を容易に理解することが困難となっており、プログラムの誤りを生じさせる虞があるばかりでなく、正誤検出にも時間を要するという問題が生じていた。

【０００６】

また、機能が複雑化しているが故に、その機能をソフトウェア上で実現するかハードウェア上で実現するかの判断も困難となっており、これらが容易に交換可能となる仕組みが必要となっているのが実情である。

【０００７】

一方、近年では、インターフェース定義言語（ＩＤＬ＝Ｉｎｔｅｒｆａｃｅ Ｄｅｆｉｎｉｔｉｏｎ Ｌａｎｇｕａｇｅ）によりソフトウェアオブジェクトとのインターフェースを定義して活用する手法が普及している（例えば、ＯＭＧ（Ｏｂｊｅｃｔ Ｍａｎａｇｅｍｅｎｔ Ｇｒｏｕｐ）によるＣＯＲＢＡのＩＤＬや、マイクロソフト社のＣＯＭ（Ｃｏｍｐｏｎｅｎｔ Ｏｂｊｅｃｔ Ｍｏｄｅｌ）のＩＤＬが知られている）。

【０００８】

しかしながら、これらはソフトウェアオブジェクトを対象としていて、ハードウェアオブジェクト（装置）とのインターフェースは考慮されていないという問題が生じている。

【０００９】

そこで、本願出願人は、これらのインターフェース定義言語を用いてハードウェアとのインターフェースを定義することができれば、ソフトウェアとハードウェアを統一的に扱うことができ、よってソフトウェアオブジェクトとハードウェアオブジェクトを容易に交換することが可能になるなど、プログラマーの負担を

著しく減少させることができ、しかも、ソフトウェアとのインターフェース方法の定義がより簡単なものとなってハードウェア設計者の負担も減少させることができるという考えに至った。

【００１０】

本発明は、上述のような従来の問題に鑑みてなされたもので、周辺ＬＳＩのような論理回路により構成される装置のインターフェースを改善し、プログラマーがレジスタの意味や多数のレジスタ間の関連性を理解するための負担を軽減することができ、しかも、ソフトウェアオブジェクトとハードウェアオブジェクトとの扱いを統一可能とし得て、ソフトウェアオブジェクトとハードウェアオブジェクトの交換を容易にすることができる論理回路のインターフェース方法およびインターフェースを備えた装置を提供することを目的とする。

【００１１】

【課題を解決するための手段】

その目的を達成するため、本発明の論理回路のインターフェース方法は、論理演算素子により構成される論理回路において、インターフェース定義言語を用いてインターフェースを定義し、前記インターフェース定義言語はソフトウェアオブジェクトに対するインターフェース定義言語と共通部分を有し、前記インターフェース定義言語は機能ごとに機能名および該機能に対する引数および戻り値を宣言する手段を具備し、前記インターフェースを実現するサーバーインターフェース回路には前記インターフェース定義言語に記述された前記機能名を識別する入力および引数の入出力を行う手段および戻り値の出力を行う手段のうち少なくとも前記機能名を識別する入力を行う手段を具備させることを特徴とする。

【００１２】

また、本発明の論理回路のインターフェース方法は、論理演算素子により構成される論理回路において、インターフェース定義言語を用いてインターフェースを定義し、前記インターフェース定義言語はソフトウェアオブジェクトに対するインターフェース定義言語と共通部分を有し、前記インターフェース定義言語は機能ごとに機能名および該機能に対する引数および戻り値を宣言する手段を具備し、前記インターフェースを実現するクライアントインターフェース回路には前

記インターフェース定義言語に記述された前記機能名を識別する出力および引数の出入力を行う手段および戻り値の入力を行う手段のうち少なくとも前記機能名を識別する出力を行う手段を具備させることを特徴とする。

【００１３】

また、本発明の論理回路のインターフェース方法は、論理演算素子により構成される論理回路において、インターフェース定義言語を用いてインターフェースを定義し、前記インターフェース定義言語はソフトウェアオブジェクトに対するインターフェース定義言語と共通部分を有し、前記インターフェース定義言語は機能ごとに機能名および該機能に対する引数および戻り値を宣言する手段を具備し、前記インターフェースを実現するサーバーインターフェース回路として前記インターフェース定義言語に記述された前記機能名を識別する入力および引数の入出力を行う手段および戻り値の出力を行う手段のうち少なくとも前記機能名を識別する入力を行う手段を具備したサーバー側論理回路と、前記インターフェースを実現するクライアントインターフェース回路として前記機能名を識別する出力および引数の出入力を行う手段および戻り値の入力を行う手段のうち少なくとも前記機能名を識別する出力を行う手段を具備するクライアント側論理回路とを備え、前記サーバー側論理回路と前記クライアント側論理回路とに関し、前記機能名を識別する出力を行う手段から前記機能名を識別する入力を行う手段へのデータ転送を可能とし、且つ、前記引数を入出力する手段と前記引数を出入力する手段とが互いに備わっていれば前記引数を入出力する手段と前記引数を出入力する手段とのデータ転送を可能とし、且つ、前記戻り値を出力する手段と前記戻り値を入力する手段とが互いに備わっていれば前記戻り値を出力する手段から前記戻り値を入力する手段へのデータ転送を可能とする論理回路を合成することを可能とすることを特徴とする。

【００１４】

さらに、本発明のインターフェースを備えた装置は、論理演算素子により構成される論理回路において、インターフェース定義言語を用いてインターフェースを定義し、前記インターフェース定義言語はソフトウェアオブジェクトに対するインターフェース定義言語と共通部分を有し、前記インターフェース定義言語は

機能ごとに機能名および該機能に対する引数および戻り値を宣言する手段を具備し、前記インターフェースを実現するサーバーインターフェース回路には前記インターフェース定義言語に記述された前記機能名を識別する入力および引数の入出力を行う手段および戻り値の出力を行う手段のうち、少なくとも前記機能名を識別する入力を行う手段を具備することを特徴とする。

【００１５】

また、本発明のインターフェースを備えた装置は、論理回路において、インターフェース定義言語を用いてインターフェースを定義し、前記インターフェース定義言語はソフトウェアオブジェクトに対するインターフェース定義言語と共通部分を有し、前記インターフェース定義言語は機能ごとに機能名および該機能に対する引数および戻り値を宣言する手段を具備し、前記インターフェースを実現するクライアントインターフェース回路には前記インターフェース定義言語に記述された前記機能名を識別する出力および引数の入出力を行う手段および戻り値の入力を行う手段のうち、少なくとも前記機能名を識別する出力を行う手段を具備することを特徴とする。

【００１６】

また、前記クライアントインターフェース回路の接続端子は、前記サーバーインターフェース回路の接続端子とシステムバスとのいずれとも接続可能とし、前記サーバーインターフェース回路の接続端子へ接続した場合には前記サーバーインターフェース回路を介して装置を駆動可能とし、システムバスに接続した場合には中央処理装置は前記クライアントインターフェース内部のレジスタの値を読み書き可能とし、前期中央処理装置は前記サーバーインターフェース回路を介する装置の機能を代替することを可能とするインターフェースを具備することを特徴とする。

【００１７】

【発明の実施の形態】

次に、本発明の実施の形態を図面に基づいて説明する。

【００１８】

図１は、本発明のコンピュータシステムの構成図である。この図１において、

１はＣＰＵ（中央処理装置）、２はメモリ、３は乱数発生装置である。このメモリ２と乱数発生装置３とはシステムバス４を介してＣＰＵ１と接続されている。また、ＣＰＵ１はメモリ２上に記憶されるプログラムを解釈・処理する。

【００１９】

図２は乱数発生装置３のブロック構成図である。乱数発生装置３は、ＩＤＬサーバーインターフェース部１１を有する。

【００２０】

このＩＤＬサーバーインターフェース部１１は、機能識別レジスタ１２、引数レジスタ１３、戻り値レジスタ１４、状態レジスタ１５を備え、外部端子１６によりシステムバス４に接続される。尚、図２では、各レジスタ１２～１５とシステムバス４とのデータの授受を制御する部分は省略されているが、システムバス４に含まれるアドレスバスや制御バスにより各レジスタ１２～１５とのデータの受渡しが制御される。

【００２１】

図３はＩＤＬによるインターフェースの定義例の説明図である。この例は、上述したＣＯＲＢＡのＩＤＬとの共通部分があり、インターフェースの名称、機能名、機能名ごとの引数および戻り値が記してある。また、図３の例では、インターフェース名をｒａｎｄｏｍＧｅｎｅｒａｔｏｒとする乱数発生装置３のインターフェースを定義したもので、乱数の種を設定する機能（機能名：ｓｅｔＳｅｅｄ）と、発生した乱数を得る機能（機能名：ｇｅｔＲａｎｄｏｍ）が示されている。そして、ｓｅｔＳｅｅｄには入力引数としてｄｏｕｂｌｅ型の種（ｓｅｅｄ）が渡されることを表している。また、ｇｅｔＲａｎｄｏｍは戻り値として発生した乱数をｄｏｕｂｌｅ型の値として返すことを表している。

【００２２】

図４および図５は、図３に例示したインターフェース定義を元に、本発明によるインターフェースを備えた装置をＣＰＵ１により操作するために作成したＣ言語のヘッダファイルおよびプログラムの説明図である。

【００２３】

このようなヘッダファイルやプログラムは、インターフェース定義から一定の

規則に基づいて変換して作成できるため、通常ＩＤＬコンパイラと呼ばれるプログラムにより変換を行う。ＣＯＲＢＡなどのＩＤＬコンパイラはインターフェース定義を通信データに変換するＣやＣ＋＋などのヘッダファイルやプログラムを生成するが、ここでは乱数発生装置３のＩＤＬサーバーインターフェース部１１のレジスタ設定または参照を行うＣのプログラムが生成されるよう、ＩＤＬコンパイラが構成される。なお、このような役割を果たすプログラムは、スタブあるいはプロキシなどと呼ばれる。

【００２４】

図５では、インターフェース定義されたｓｅｔＳｅｅｄとｇｅｔＲａｎｄｏｍに対応する関数『ｓｅｔＳｅｅｄ』と『ｇｅｔＲａｎｄｏｍ』とが記述されている。

【００２５】

本発明の方法あるいは装置は複数部の協調により達成される。図１および図２に示した構成図をもとに、これらの関数および乱数発生装置３の動作の概要を説明する。

【００２６】

ＣＰＵ１はメモリ２に格納されているプログラムを読み出して処理を行う。そのプログラム内で関数『ｓｅｔＳｅｅｄ』が呼び出されると、関数『ｓｅｔＳｅｅｄ』のプログラムに基づき、ＣＰＵ１は機能識別の値『ＦＩＤ＿ｓｅｔＳｅｅｄ』を乱数発生装置３の機能識別レジスタ１２に書き込む。

【００２７】

この定数『ＦＩＤ＿ｓｅｔＳｅｅｄ』は図４のヘッダファイルにて定義されているもので、以下の『ＦＩＤ＿ｇｅｔＲａｎｄｏｍ』や『Ｆｉｎｉｓｈｅｄ』などの定数も同様である。続いてｄｏｕｂｌｅ型の引数『ｓｅｅｄ』を引数レジスタ１３に書き込み、この関数の処理は終了する。

【００２８】

乱数発生装置３のＩＤＬサーバーインターフェース部１１は、機能ｓｅｔＳｅｅｄに対するすべての引数を受け取ったことを確認すると、直ちに動作を開始させる。この例の場合、乱数発生部１７の乱数の種レジスタ１８に乱数の種を格納

する動作が行われる。

【００２９】

次に関数『ｇｅｔＲａｎｄｏｍ』が呼び出されると、機能識別の値『ＦＩＤ＿ｇｅｔＲａｎｄｏｍ』を乱数発生装置３の機能識別レジスタ１２に書き込む。ＩＤＬサーバーインターフェース部１１は、機能『ｇｅｔＲａｎｄｏｍ』に引数がないことから、直ちに乱数発生部に乱数の発生をさせる。ＩＤＬサーバーインターフェース部１１は乱数発生部１７の乱数の発生が完了すると、それを戻り値レジスタ１４に書き込む。関数『ｇｅｔＲａｎｄｏｍ』では、戻り値レジスタ１４に値が書き込まれるまで、関数『ｇｅｔＳｔａｔｕｓ』を呼び出して状態レジスタ１５の値を読み取り、その値が『Ｆｉｎｉｓｈｅｄ』となるまでｗｈｉｌｅ文により待ち続ける。以上、本発明の装置を組み込んだコンピュータシステムの全体の動作の概要を説明した。

【００３０】

次に、図７のフロー図を用いて図１に示したコンピュータシステムの設計作業の流れを説明する。尚、以下のフロー図並びにその説明では、各ステップを記号「Ｓ」で省略する。

【００３１】

設計作業は、最初にインターフェースを決定する（Ｓ２１）。インターフェースの決定では機能をすべて挙げ、各機能ごとに機能名、引数、戻り値をそれぞれ決定する。

【００３２】

次に、Ｓ２１で決定したインターフェースを図３のインターフェース定義ファイルとして作成する（Ｓ２２）。そして、インターフェース定義ファイルをＩＤＬコンパイラに通してヘッダファイルとスタブを作成する（Ｓ２３）。ＩＤＬコンパイラにより図４のヘッダファイルと図５のスタブが生成される。

【００３３】

ここでソフトウェア設計とハードウェア設計に分かれる。

【００３４】

ソフトウェア設計では、次にスタブをコンパイラーによりコンパイルしてリロ

ケータブルオブジェクトを作成する（Ｓ２４）。次に他の部分のリロケータブルオブジェクトとリンクして実行モジュールを作成し（Ｓ２５）、ソフトウェアの設計作業は完了する。

【００３５】

ハードウェア設計者は、図３のインターフェース定義ファイルと図４のヘッダファイルを参照しながらＩＤＬサーバーインターフェース部１１を設計する（Ｓ２６）。ここで参照するインターフェース定義ファイルとヘッダファイルの参照する内容については、ＩＤＬサーバーインターフェース部１１の動作の詳細とともに説明する。

【００３６】

次にハードウェア設計者は、乱数発生部１７の回路とＩＤＬサーバーインターフェース部１１を結合する回路を設計して（Ｓ２７）、乱数発生装置３の設計作業は完了する。

【００３７】

引き続き、ＣＰＵ１、メモリ２、乱数発生装置３，システムバス４を接続する回路の設計作業を行い（Ｓ２８）、ハードウェア設計作業は終了する。

【００３８】

次に、図６のＩＤＬサーバーインターフェース部１１の動作を表すフローチャートを用いて、本発明の中心であるＩＤＬサーバーインターフェース部１１の動作を、もう一度詳しく説明する。

【００３９】

ＩＤＬサーバーインターフェース部１１は、機能識別レジスタ１２に外部端子１６を介して値が書き込まれるのを待つ（Ｓ１）。

【００４０】

ここで、機能識別レジスタ１２に書き込まれる値は、ｓｅｔＳｅｅｄ機能であれば、これに対応する値は、図４のヘッダファイルにおいて＃ｄｅｆｉｎｅにより定義される『ＦＩＤ＿ｓｅｔＳｅｅｄ』の値であり、具体的な値は『１』である。また、ｇｅｔＲａｎｄｏｍ機能であれば『ＦＩＤ＿ｇｅｔＲａｎｄｏｍ』であり、具体的な値は『２』である。

【００４１】

機能識別レジスタ１２に値が書き込まれると、状態レジスタ１５の値を実行中を意味する値『Ｅｘｅｃｕｔｉｎｇ』に変更し（Ｓ２）、機能識別レジスタ１２の内容から引数の個数を割り出す（Ｓ３）。ここで、引数の個数の割り出しは、引数個数算出部２０により行われるが、引数個数算出部２０には図３のインターフェース定義ファイルおよび図４のヘッダを参照して、予め機能識別と引数個数との対応関係を引数個数算出部２０に記憶するように設計しておく。

【００４２】

次に、引数がすべて入力されているかどうかを判定し（Ｓ４）、受け取っていない引数があれば引数レジスタ１３に引数が１つ書き込まれるのを待ち（Ｓ５）、再びＳ４から繰り返す。もしすべての引数が入力されていれば（引数がない場合は、すべて入力されたものとみなす）、機能識別により指定された機能を開始させる（Ｓ６）。ＩＤＬサーバーインターフェース部１１の設計者は、図４のヘッダファイルを参照して機能識別と対応する機能を一致させておく。

【００４３】

次に、機能処理の終了を待つ（Ｓ７）。その機能の処理が終了すると、戻り値があるかどうかを判定して（Ｓ８）、戻り値があれば戻り値を戻り値レジスタ１４に書き込んで（Ｓ９）、最後に状態レジスタ１５の値を『Ｆｉｎｉｓｈｅｄ』に変更し、再びＳ１から繰り返す。

【００４４】

以上の説明において、図５では、状態レジスタが『Ｆｉｎｉｓｈｅｄ』に変化するまで、ｗｈｉｌｅループで待つようになっているが、状態レジスタ１５の値が変化したときにＣＰＵ１へ割り込みを発生するようにして、それまでこのプログラムが待機するように変更すればＣＰＵ１の利用効率を挙げることができる。また、引数の個数やデータ型の検査を加えるなどの変更を行ってもよい。また、インターフェース定義において、すべての機能が引数を持たないのであれば、引数レジスタ１３を省略し、図６で示したルーチンのＳ４，Ｓ５を省略することもできる。また、全ての機能が戻り値を返さないのであれば、戻り値レジスタ１４および図６で示したルーチンのＳ８，Ｓ９を省略することもできる。また、乱数

発生装置３の働きをソフトウェアによりエミュレートするように変更する場合、図５の関数『ｓｅｔＳｅｅｄ』および関数『ｇｅｔＲａｎｄｏｍ』を置き換えればよく、ハードウェアオブジェクトを容易にソフトウェアオブジェクトに交換することが可能である。これは、本発明による方法および装置で用いるインターフェース定義言語がソフトウェアオブジェクトを対象とするインターフェース定義言語と共通部分があるために、容易に交換が可能となる。

【００４５】

次に、図２に示した乱数発生装置３を利用して、モンテカルロ法により円周率を計算する円周率計算装置５の構成図を図１０に示す。また、この円周率計算装置（ｐｉＣａｌｃｕｌａｔｏｒ）のインターフェース定義を図９に示す。この円周率計算装置５を組み込んだコンピュータシステムの構成図を図８に示す。

【００４６】

図１０の円周率計算装置５は、ＩＤＬサーバーインターフェース部２１、円周率計算部２７、ＩＤＬクライアントインターフェース部３１からなり、ＩＤＬサーバーインターフェース部２１に接続された外部端子２６はシステムバス４に接続され、ＩＤＬクライアントインターフェース部３１に接続された外部端子３６は乱数発生装置３の外部端子１６に接続される。

【００４７】

外部端子３６と外部端子１６の接続によって、機能識別レジスタ３２から機能識別レジスタ１２へのデータ転送、引数レジスタ３３と引数レジスタ１３とのデータ転送、戻り値レジスタ１４から戻り値レジスタ３４へのデータ転送が可能となり、円周率計算装置５から乱数発生装置３に対する機能の要求が可能となる。

【００４８】

図８のコンピュータシステムにおいて、乱数発生装置３を使用しないように変更したコンピュータシステムの構成図が図１１である。図１１では、円周率計算装置５のＩＤＬクライアントインターフェース部３１に接続された外部端子３６はシステムバス４に接続される。

【００４９】

コンピュータシステムを図８の構成から図１０の構成に変更可能とするために

、円周率発生装置５のＩＤＬクライアントインターフェース部３１は、乱数発生装置３のＩＤＬサーバーインターフェース部１１を駆動可能とするとともに、ＣＰＵ１から機能識別レジスタ３２、引数レジスタ３３、戻り値レジスタ３４、状態レジスタ３５を読み出しまたは書き込み可能とすることで達成される。

【００５０】

この時、ＣＰＵ１は、円周率発生装置５のＩＤＬクライアントインターフェース部３１から発せされる要求を監視して、機能識別に応じた機能の処理を行う必要がある。

【００５１】

図１２は、ＣＰＵ１が行う、要求の監視、機能の処理を説明するフローチャートである。

【００５２】

先ず、円周率計算装置５のＩＤＬクライアントインターフェース部３１の状態レジスタ３５の値を読み出す（Ｓ３１）。その値が『Ｒｅｑｕｅｓｔｉｎｇ』であれば、機能の処理の要求があると判断し（Ｓ３２）、機能識別レジスタ３２の値を読み出す（Ｓ３３）。

【００５３】

機能識別レジスタ３２の値が『ＦＩＤ＿ｓｅｔＳｅｅｄ』であれば（Ｓ３４）、引数レジスタ３３の値を読み出し、乱数の種として記憶する（Ｓ３５）。

【００５４】

機能識別レジスタ３２の値が「ＦＩＤ＿ｇｅｔＲａｎｄｏｍ」であれば（Ｓ３６）、乱数を発生し（Ｓ３７）、その乱数を戻り値レジスタ３４へ書き込む（Ｓ３８）。

【００５５】

上述したＳ３５、Ｓ３８が終了すれば、再びＳ３１へ戻る。このプログラムはＣＰＵ１を占有してしまうので、タスクの１つとして実装する必要がある。あるいは、状態レジスタ３５の値が『Ｒｅｑｕｅｓｔｉｎｇ』に変化したときに、ＣＰＵ１に割り込みが発生するようにして、それまでＳ３１で待機するようにすれば、ＣＰＵ１の利用効率を向上することができる。

【００５６】

図１３、図１４は図１２の処理をＣ言語化したヘッダファイルおよびプログラムである。インターフェース定義から一定の規則に従って変換できるため、先述の図４、図５のヘッダファイル、プログラムファイルと同様にＩＤＬコンパイラと呼ばれるプログラムを使って作成することができる。なお、このような役割を果たすプログラムは、スケルトンあるいはスタブなどと呼ばれる。

【００５７】

図１５は、上述のような特性を持ったＩＤＬクライアントインターフェース部３１の動作を表すフローチャートである。始めに円周率計算部２７から機能要求がないかを判断し（Ｓ４１）、もしあれば機能識別レジスタ３２に機能要求に対応する機能識別を書き込む（Ｓ４２）。ここでは、『ＦＩＤ＿ｓｅｔＳｅｅｄ』か『ＦＩＤ＿ｇｅｔＲａｎｄｏｍ』のいずれかの値が書き込まれる。次に機能要求がｓｅｔＳｅｅｄであるかを判断し（Ｓ４３）、ｓｅｔＳｅｅｄであれば引数が存在するので、円周率計算部２７から引数を受け取り引数レジスタ３３に書き込み（Ｓ４４）、状態レジスタ３５の値を『Ｒｅｑｕｅｓｔｉｎｇ』へと変更する（Ｓ４５）。

【００５８】

ここで円周率計算装置５の外部端子３６の接続先が乱数発生装置３であるか、ＣＰＵ１であるかにより動作が変わる。

【００５９】

接続先がＣＰＵ１であれば、ＣＰＵ１は先述の図１２のフローチャートで示した処理を行っており、機能識別の値が『ＦＩＤ＿ｓｅｔＳｅｅｄ』であれば、引数はＳ３５で読み込まれる。機能識別の値が『ＦＩＤ＿ｇｅｔＲａｎｄｏｍ』であれば、引数はない。Ｓ４７では、すべての引数が読み込まれたかどうかを判断し（引数がない場合は、読み込まれたものと判断する）、すべて読み込まれている場合は、戻り値があるかどうか判断し（Ｓ４８）、戻り値がある場合は戻り値レジスタ３４に値が書き込まれるまで待つ（Ｓ４９）。戻り値レジスタ３４に書き込まれた値を乱数として円周率計算部２７に渡す（Ｓ５０）。

【００６０】

以上でＣＰＵ１とのやり取りは終わって、状態レジスタ３５の値を『Ｗａｉｔｉｎｇ』に変更し（Ｓ５１）、再びＳ４１から繰り返す。

【００６１】

もし、Ｓ４６にて、円周率計算装置５の外部端子３６の接続先が乱数発生装置３であると判断された場合には、機能識別レジスタ３２の内容を接続先である乱数発生装置３のＩＤＬサーバーインターフェース部１１の機能識別レジスタ１２に書き込む（Ｓ５２）。

【００６２】

ここで乱数発生装置３は図６のフローチャートに従って動作しており、Ｓ１において機能識別レジスタ１２への書き込みを検出して、Ｓ２以降の動作を行う。ＩＤＬクライアントインターフェース部３１のＳ５３以降の動作は、乱数発生装置３のＩＤＬサーバーインターフェース部１１のＳ４以降の動作と対応付けられている。

【００６３】

次に、引数があるかどうかを判断する（Ｓ５３）。機能識別の値が『ＦＩＤ＿ｓｅｔＳｅｅｄ』であれば、引数があると判断される。引数があれば引数レジスタ３３の内容を乱数発生装置３の引数レジスタ１３に書き込む（Ｓ５４）。

【００６４】

次に、戻り値があるかどうかを判断する（Ｓ５５）。機能識別の値が『ＦＩＤ＿ｇｅｔＲａｎｄｏｍ』であれば戻り値があると判断される。戻り値があると判断された場合、乱数発生装置３の状態レジスタ１５の値が『Ｆｉｎｉｓｈｅｄ』になるまで待つ（Ｓ５６）。続いて乱数発生装置３の戻り値レジスタ１４の値を読み出し、戻り値レジスタ３４に書き込む（Ｓ５７）。そして、戻り値レジスタ３４の内容を円周率計算部２７に渡す（Ｓ５８）。

【００６５】

以上で乱数発生装置３とのやり取りは終わって、状態レジスタ３５の値を『Ｗａｉｔｉｎｇ』に変更し（Ｓ５１）、再びＳ４１から繰り返す。

【００６６】

ところで、上記実施の形態では、乱数発生装置３と円周率計算装置５を例に取

って説明したが、乱数発生部１７や、円周率計算部２７を変更し、またそれにあわせてインターフェース定義（ｒａｎｄｏｍＧｅｎｅｒａｔｏｒ，ｐｉＣａｌｃｕｌａｔｏｒ）を変更することにより、種々の装置に対応できる。

【００６７】

また、上記実施の形態では、引数の型や引数の個数の照合を行っていないが、それらを行うことにより、インターフェースプログラムの誤りを厳密に検出することができる。

【００６８】

さらに、上記実施の形態では、入力引数の例のみを示したが出力引数あるいは入出力引数の扱いは可能で、出力引数の場合、入力引数の場合とはデータ転送の向きが逆になる。また、本実施例ではインターフェース記述言語としてＣＯＲＢＡのＩＤＬを例に説明したが、ソフトウェアオブジェクトを対象とするＩＤＬと共通部分があり、機能名および該機能に対する引数および戻り値を宣言する手段を備えたものであればＣＯＲＢＡのＩＤＬに限定されるものではない。

【００６９】

また、上記実施の形態のプログラムではインスタンスオブジェクトを区別するための記述を省略して簡略化しているが、インスタンスオブジェクトを区別して取り扱う手段を備えることが望ましい。

【００７０】

尚、本発明の方法及び装置は、集積回路であって外部とのインターフェースを目的とするのがもっとも効果的であるが、集積回路内部のブロック間のインターフェースあるいは複数の集積回路からなる論理回路間のインターフェースに用いてもよい。

【００７１】

【発明の効果】

本発明の方法および装置によれば、ソフトウェアオブジェクトと共通部のあるインターフェース定義言語によりハードウェアとのインターフェースを定義することができるから、ソフトウェアとハードウェアを統一的に扱うことが可能になり、ソフトウェアオブジェクトとハードウェアオブジェクト（装置）を容易に交

換することが可能になるなどプログラマーの負担を著しく減少させることができる。また、ソフトウェアとのインターフェース方法の定義がより簡単なものになり、ハードウェア設計者の負担を減少することができる。また、ハードウェアとソフトウェアの並行開発が容易になり、開発納期の短縮を図ることができる。ハードウェアオブジェクト同士の接続と中央処理装置への接続とを容易に変更できるので、ハードウェア処理していたものをソフトウェア処理に置き換える、あるいはその逆が容易になるなど絶大な効果を有する。

【図面の簡単な説明】

【図１】　本発明のコンピュータシステムの構成図である。

【図２】　乱数発生装置のブロック構成図である。

【図３】　インターフェース定義言語（ＩＤＬ）による乱数発生装置のインターフェース定義の説明図である。

【図４】　図５のプログラムに対応するヘッダファイルの説明図である。

【図５】　乱数発生装置を直接操作するためのプログラムの説明図である。

【図６】　ＩＤＬサーバーインターフェース部の動作を表すフロー図である。

【図７】　設計作業の流れを表すフロー図である。

【図８】　円周率計算装置を組み込んだコンピュータシステムのブロック構成図である。

【図９】　円周率計算装置（ｐｉＣａｌｃｕｌａｔｏｒ）のインターフェース定義の説明図である。

【図１０】　円周率を計算する円周率計算装置のブロック構成図である。

【図１１】　乱数発生装置を使用しないように変更したコンピュータシステムのブロック構成図である。

【図１２】　中央処理装置が行う、要求の監視及び機能の処理に関するフロー図である。

【図１３】　図１４のプログラムに対応するヘッダファイルの説明図である。

【図１４】　図１２の処理をＣ言語化したプログラムの説明図である。

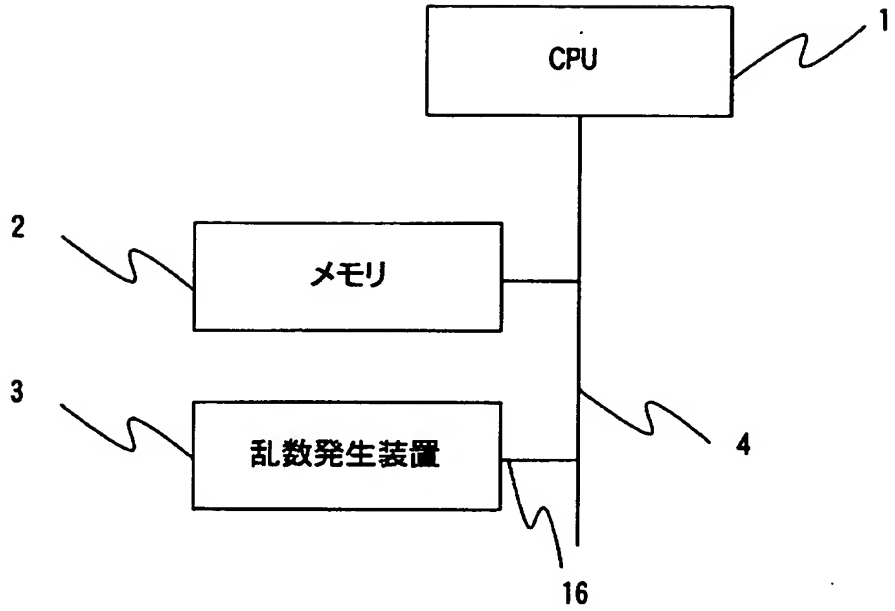
【図１５】　ＩＤＬクライアントインターフェース部の動作を表すフロー図である。

【符号の説明】

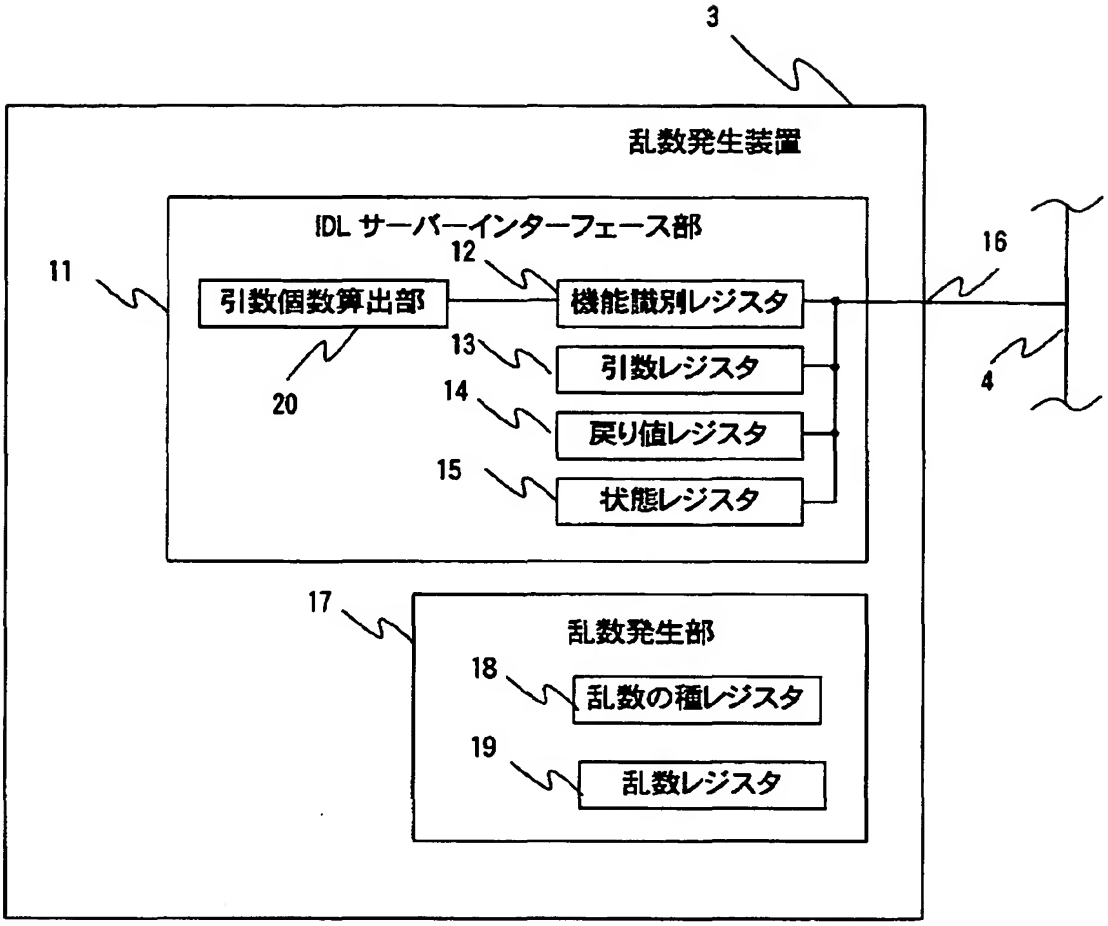
１　中央制御装置、２　メモリ、３　乱数発生装置、４　システムバス、５　円周率計算装置、１１　（乱数発生装置の）ＩＤＬサーバーインターフェース部、１２　機能識別レジスタ、１３　引数レジスタ、１４　戻り値レジスタ、１５　状態レジスタ、１６　外部端子、１７　乱数発生部、１８　乱数の種レジスタ、１９　乱数レジスタ、２１　（円周率計算装置の）ＩＤＬサーバーインターフェース部、２２　機能識別レジスタ、２３　引数レジスタ、２４　戻り値レジスタ、２５　状態レジスタ、２６　外部端子、２７　円周率計算部、２８　回数レジスタ、２９　円周率レジスタ、３１　ＩＤＬクライアントインターフェース部、３２　機能識別レジスタ、３３　引数レジスタ、３４　戻り値レジスタ、３５　状態レジスタ、３６　外部端子。

【書類名】　　図面

【図１】



【図２】



【図３】

```
interface randomGenerator{
    void  setSeed( in double seed  );
    double getRandom();
};
```

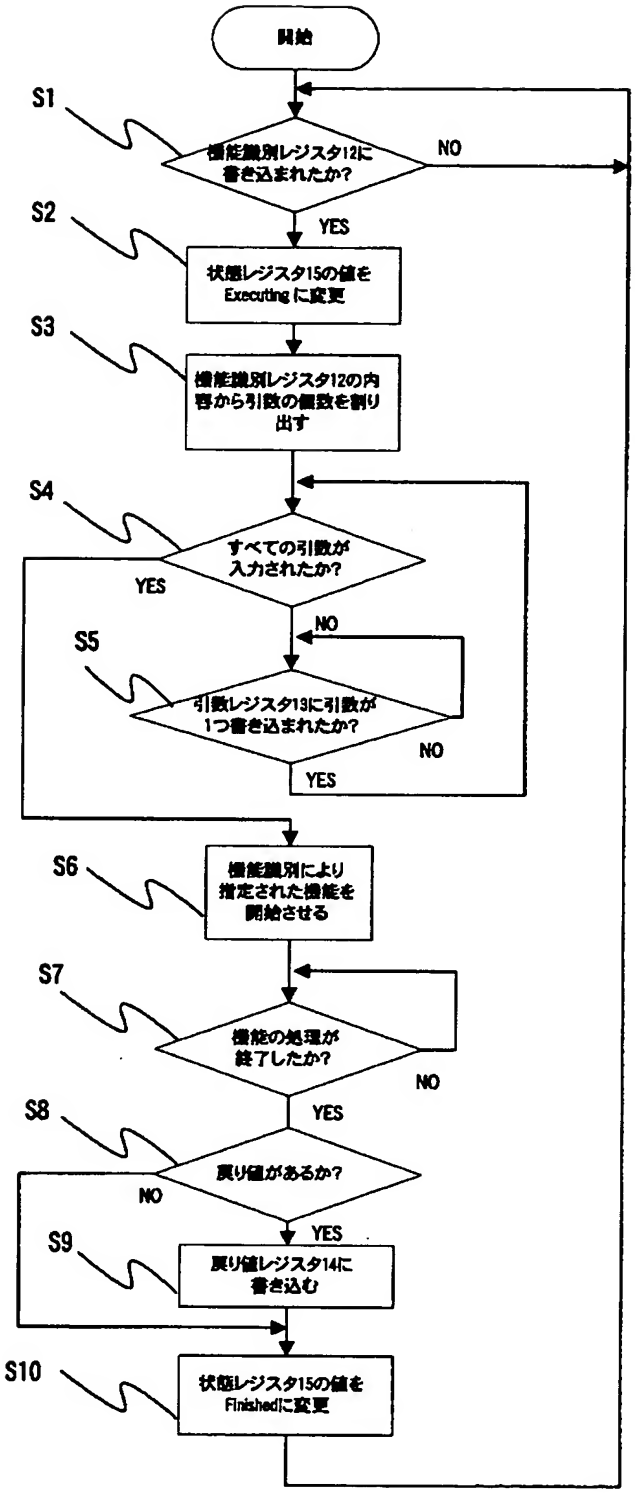
【図４】

```
/* IDL インターフェースに共通の部分 */
/* 状態レジスタの値 */
#define   Executing   1
#define   Finished    2
#define   Requesting  3

/* レジスタにアクセスする関数 */
void      putFunctionID( FID );
void      putDoubleArg( double );
double    getDoubleReturn();
int       getStatus();

/* randomGenerator 固有部 */
/* 機能識別の値の定義 */
#define  FID_setSeed      1
#define  FID_getRandom    2

/* 機能に対応する C 言語関数のプロトタイプ宣言 */
void     setSeed( double seed );
double   getRandom( )
```

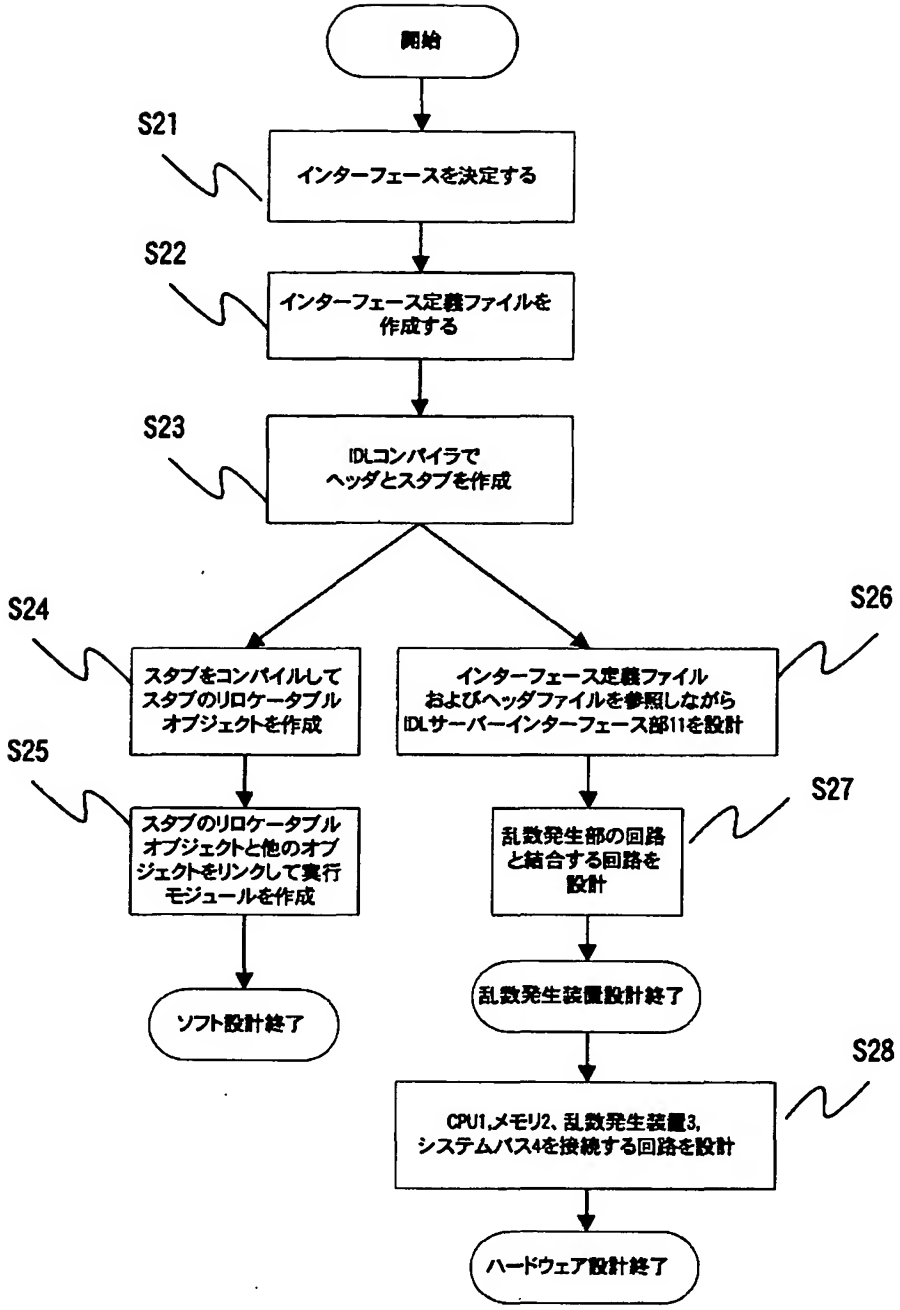
【図５】

```
void  setSeed( double seed )
{
    putFunctionID( FID_setSeed );
    putDoubleArg( seed );
}

double  getRandom( )
{
    putFunctionID( FID_getRandom );
    while( getStatus() != Finished )
        ;
    return getDoubleReturn( );
}
```

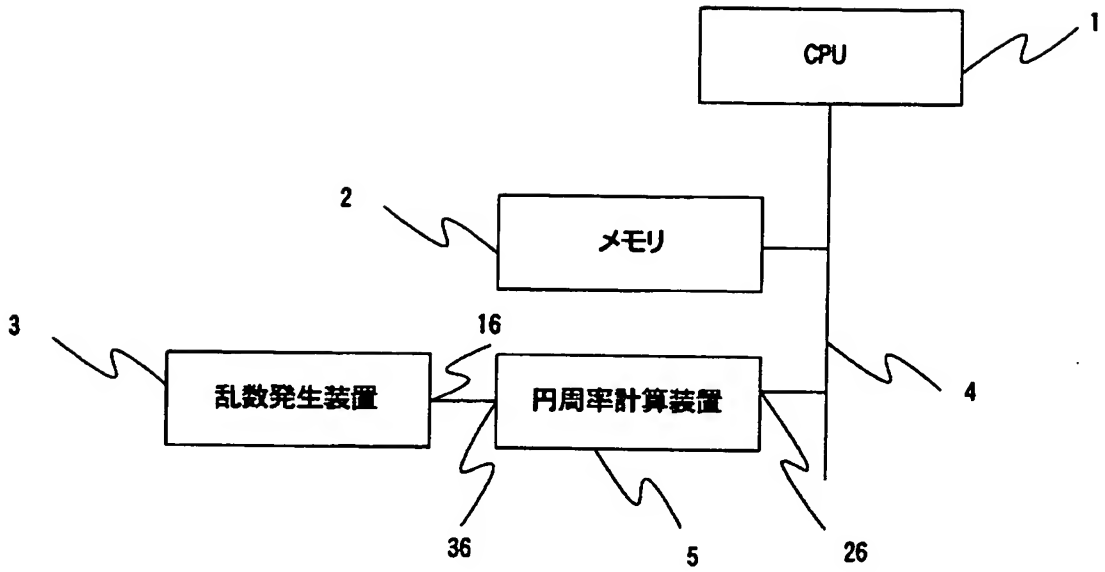
【図６】


開始
S1
機能識別レジスタ12に書き込まれたか?
NO
YES
S2
状態レジスタ15の値をExecutingに変更
S3
機能識別レジスタ12の内容から引数の個数を割り出す
S4
すべての引数が入力されたか?
YES
NO
S5
引数レジスタ13に引数が1つ書き込まれたか?
NO
YES
S6
機能識別により指定された機能を開始させる
S7
機能の処理が終了したか?
NO
YES
S8
戻り値があるか?
NO
YES
S9
戻り値レジスタ14に書き込む
S10
状態レジスタ15の値をFinishedに変更

【図７】


開始
S21
インターフェースを決定する
S22
インターフェース定義ファイルを作成する
S23
IDLコンパイラでヘッダとスタブを作成
S24
スタブをコンパイルしてスタブのリロケータブルオブジェクトを作成
S25
スタブのリロケータブルオブジェクトと他のオブジェクトをリンクして実行モジュールを作成
ソフト設計終了
S26
インターフェース定義ファイルおよびヘッダファイルを参照しながらIDLサーバーインターフェース部11を設計
S27
乱数発生部の回路と結合する回路を設計
乱数発生装置設計終了
S28
CPU1,メモリ2、乱数発生装置3,システムバス4を接続する回路を設計
ハードウェア設計終了

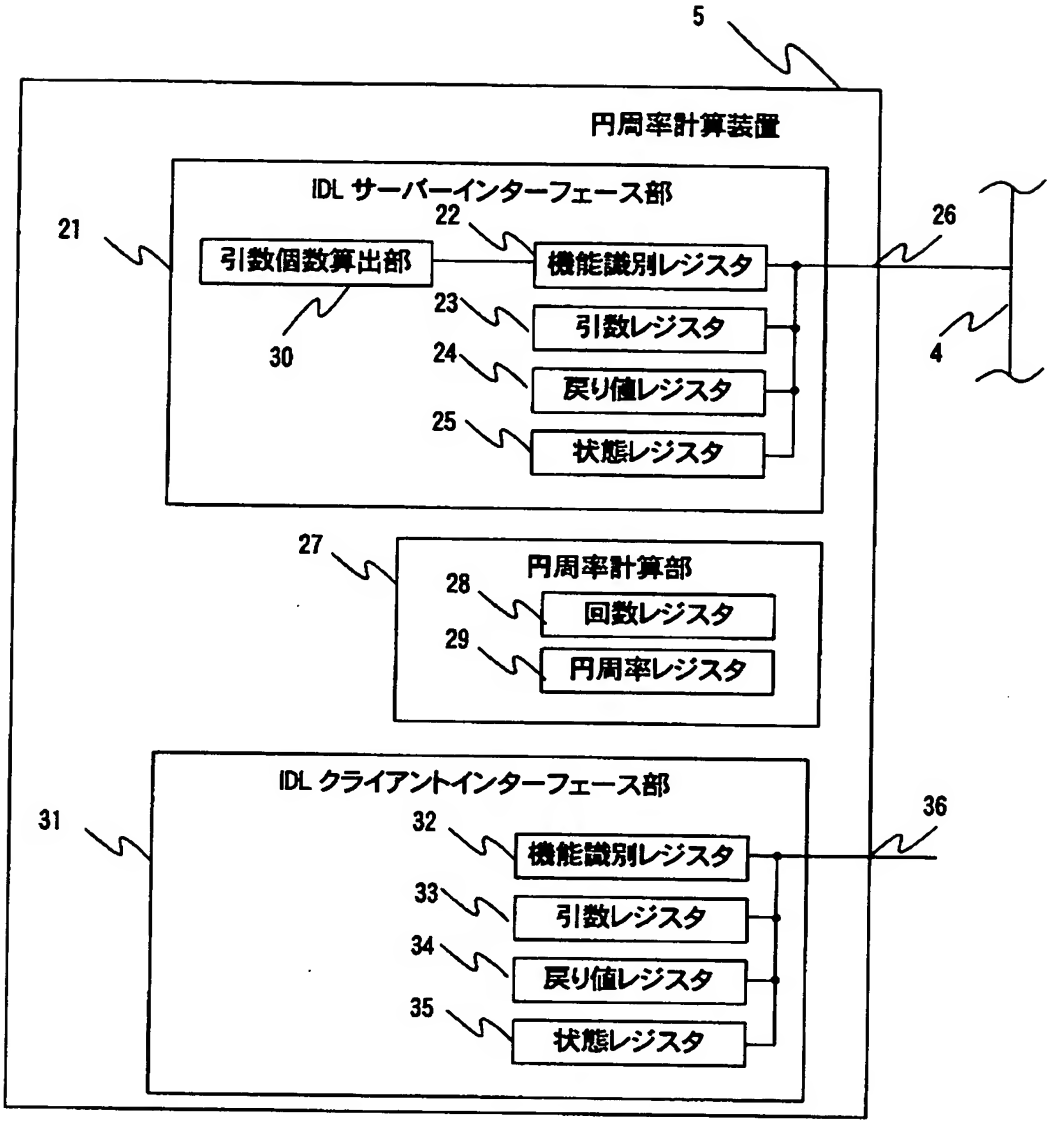
【図８】



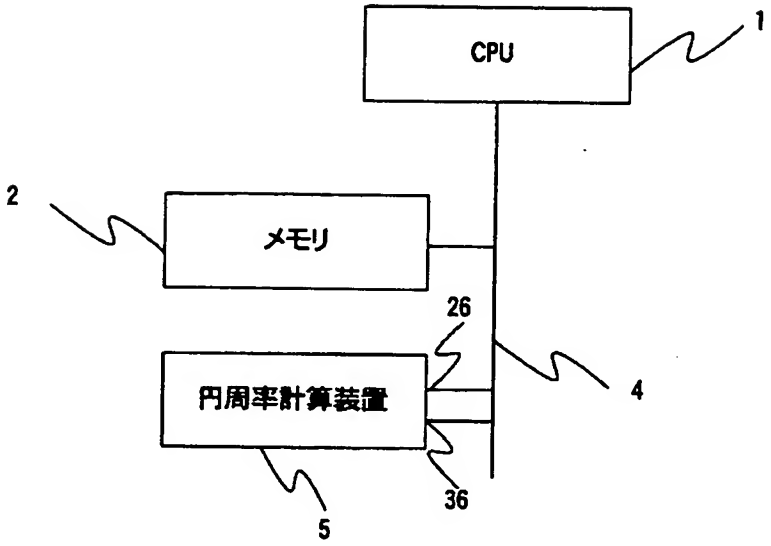
【図９】

```
interface piCalculator{
    void  setCount( in int count );
    double getPi();
};
```

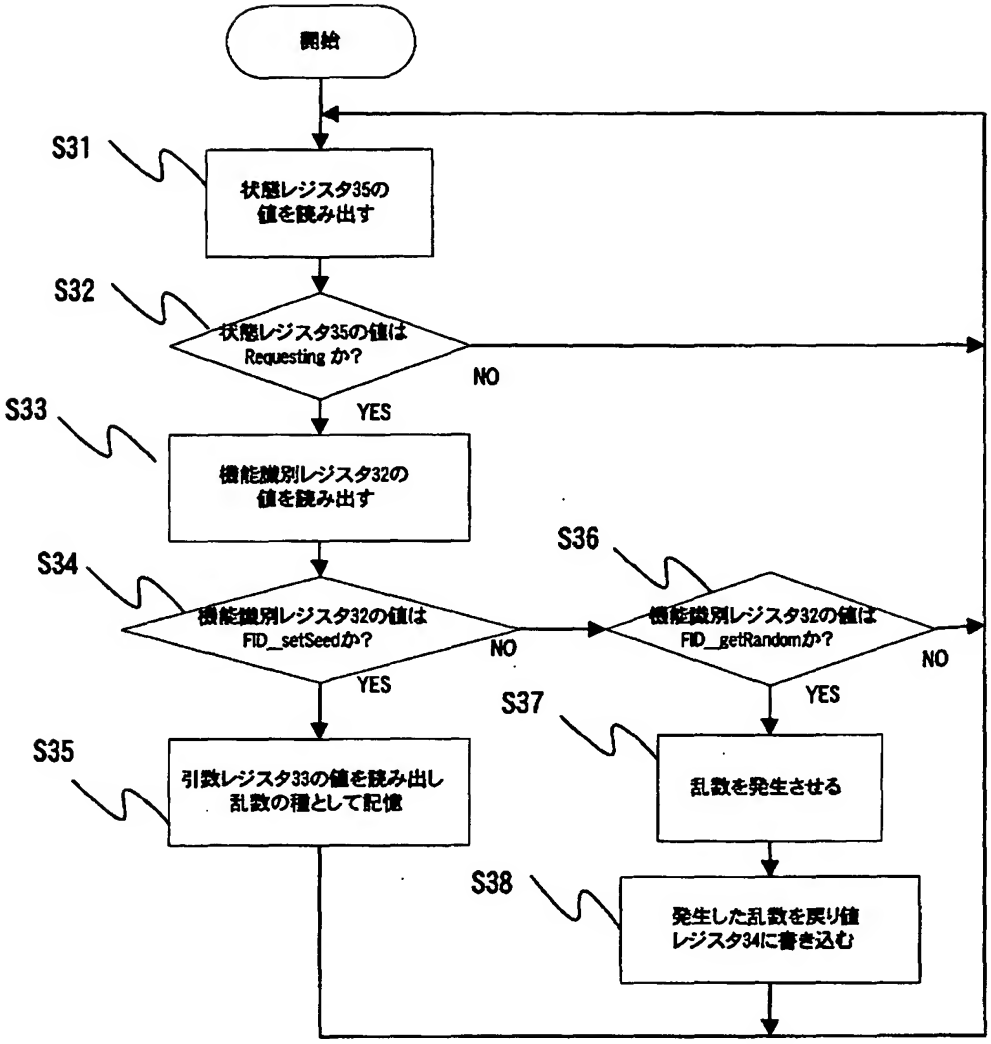
【図１０】


5
円周率計算装置
IDL サーバーインターフェース部
21
22
26
引数個数算出部
機能識別レジスタ
23
引数レジスタ
30
4
24
戻り値レジスタ
25
状態レジスタ
27
円周率計算部
28
回数レジスタ
29
円周率レジスタ
IDL クライアントインターフェース部
31
32
36
機能識別レジスタ
33
引数レジスタ
34
戻り値レジスタ
35
状態レジスタ

【図１１】



【図１２】


開始
S31
状態レジスタ35の
値を読み出す
S32
状態レジスタ35の値は
Requesting か?
NO
YES
S33
機能識別レジスタ32の
値を読み出す
S34
機能識別レジスタ32の値は
FID_setSeedか?
NO
YES
S36
機能識別レジスタ32の値は
FID_getRandomか?
NO
YES
S35
引数レジスタ33の値を読み出し
乱数の種として記憶
S37
乱数を発生させる
S38
発生した乱数を戻り値
レジスタ34に書き込む

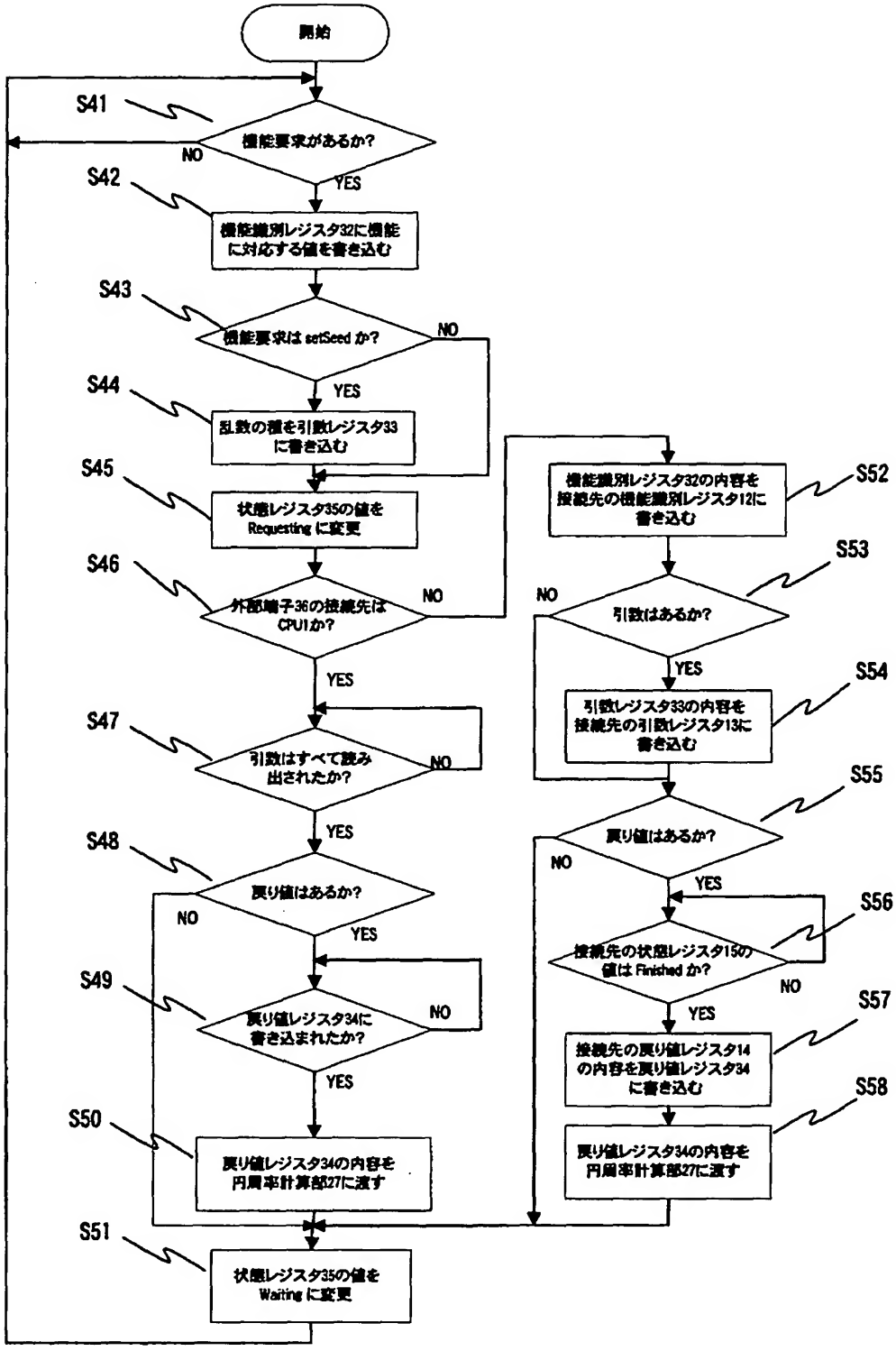
【図１３】

```
/*  IDL インターフェースに共通の部分  */
/* 状態レジスタの値 */
#define   Waiting       0
#define   Executing     1
#define   Finished      2
#define   Requesting    3

/* レジスタにアクセスする関数  */
FID       gettFunctionID();
double    getDoubleArg();
void      putDoubleReturn( double );
int       getStatus();

/* randomGenerator 固有部 */
/* 機能識別の値の定義  */
#define  FID_setSeed       1
#define  FID_getRandom     2

/* 機能に対応する C 言語関数のプロトタイプ宣言 */
void     setSeed( double seed );
double   getRandom( )
```

【図１４】

```
main()
{
    while(1){
        while( getStatus() != Requesting )
            ;
        switch( getFunctionID() ){
            case FID_setSeed:
                setSeed( getDoubleArg() );
                break;
            case FID_getRandom:
                putDoubleReturn( getRandom() );
                break;
        }
    }
}
```

【図１５】


開始
S41
機能要求があるか?
NO
YES
S42
機能識別レジスタ32に機能に対応する値を書き込む
S43
機能要求はsetSeedか?
NO
YES
S44
乱数の種を引数レジスタ33に書き込む
S45
状態レジスタ35の値をRequestingに変更
S46
外部端子36の接続先はCPU1か?
NO
YES
S47
引数はすべて読み出されたか?
NO
YES
S48
戻り値はあるか?
NO
YES
S49
戻り値レジスタ34に書き込まれたか?
NO
YES
S50
戻り値レジスタ34の内容を円周率計算部27に渡す
S51
状態レジスタ35の値をWaitingに変更
S52
機能識別レジスタ32の内容を接続先の機能識別レジスタ12に書き込む
S53
引数はあるか?
NO
YES
S54
引数レジスタ33の内容を接続先の引数レジスタ13に書き込む
S55
戻り値はあるか?
NO
YES
S56
接続先の状態レジスタ15の値はFinishedか?
NO
YES
S57
接続先の戻り値レジスタ14の内容を戻り値レジスタ34に書き込む
S58
戻り値レジスタ34の内容を円周率計算部27に渡す

【書類名】　　要約書

【要約】

【課題】　プログラマーがレジスタの意味や多数のレジスタ間の関連性を理解するための負担を軽減することができ、しかも、ソフトウェアオブジェクトとハードウェアオブジェクトとの扱いを統一可能とし、ソフトウェアオブジェクトとハードウェアオブジェクトの交換を容易にする。

【解決手段】　インターフェース定義言語を用いてインターフェースを定義し、インターフェース定義言語はソフトウェアオブジェクトに対するインターフェース定義言語と共通部分を有すると共に機能ごとに機能名および機能に対する引数および戻り値を宣言する手段を具備し、インターフェースを実現するサーバーインターフェース回路にはインターフェース定義言語に記述された機能名を識別する入力および引数の入出力を行う手段および戻り値の出力を行う手段のうち少なくとも機能名を識別する入力を行う手段を具備した。

【選択図】　　図２

出　願　人　履　歴　情　報

識別番号　　〔０００１４９０６６〕

１．変更年月日　　２００１年　７月　２日
　［変更理由］　住所変更
　　住　所　　愛知県丹羽郡大口町下小口五丁目２５番地の１
　　氏　名　　オークマ株式会社